NEC activity 1

Jan Ernée

Data preprocessing

For this whole activity, we use this https://archive.ics.uci.edu/dataset/186/wine+quality dataset. It contains data about vinho verde from Portugal. It is originally divided into two independent sets, first for the white wine and second for the red wine. For our needs, those two sets are joined to one dataset with the new categorical variable *type* with values from {red, white}. Because this is the only categorical variable we have to deal with, and also because there are just two possible values it could acquire, the one-hot transformation is used. The new dataset now contains two new columns (instead of one *type* column), one for *red* and another for *white*. The values of those columns are complementary, so if one cell contains *1* (the wine is from that class), the second cell has to be *0* (the wine cannot be in both classes at the same time). After this modification, the data has the following structure.

```
#   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         6497 non-null   float64
 1   volatile acidity      6497 non-null   float64
 2   citric acid           6497 non-null   float64
 3   residual sugar        6497 non-null   float64
 4   chlorides             6497 non-null   float64
 5   free sulfur dioxide   6497 non-null   float64
 6   total sulfur dioxide  6497 non-null   float64
 7   density               6497 non-null   float64
 8   pH                    6497 non-null   float64
 9   sulphates             6497 non-null   float64
 10  alcohol               6497 non-null   float64
 11  quality               6497 non-null   int64
 12  red                   6497 non-null   int64
 13  white                 6497 non-null   int64
```

As we can see, there are 14 columns and 6497 records. The 11th column stands for the wine quality and it is also the feature which we will be focusing on in our prediction. It is also an integer value, so the prediction thresholds are not continuous (which is necessarily not a problem for us). Other two integer columns stand for the transformed categorical variable *type*. All other columns are float numbers and represent several features which are taken to account during wine tasting.

For the preprocessing operations we use the *pandas* and *scikit-learn* libraries for python. More details about the code are in the provided .ipynb file. The data are checked for duplicates and those duplicates are removed. Check for *None* or *NaN* values is also made. Most of the data use the same units, but there are cases with the different ones (both sulfur dioxides). Regardless of the conversion, the model gives almost the same results. Also standardisation and normalisation are applied on the dataset, even if the difference between its values is quite small and also the distribution was quite similar. Those modifications still improve the performance a little bit. In the case of outliers detection, we tried two different approaches; z-score and IQR. Z-score eliminates around 600 samples, IQR around 1400. Even if the IQR removes many more samples and for the training remains just around 4000 of them, those data perform better in the final tests thus we decided to use the IQR method instead of the z-score. Finally, the quality column is placed as the last one for easier manipulation and indexing.

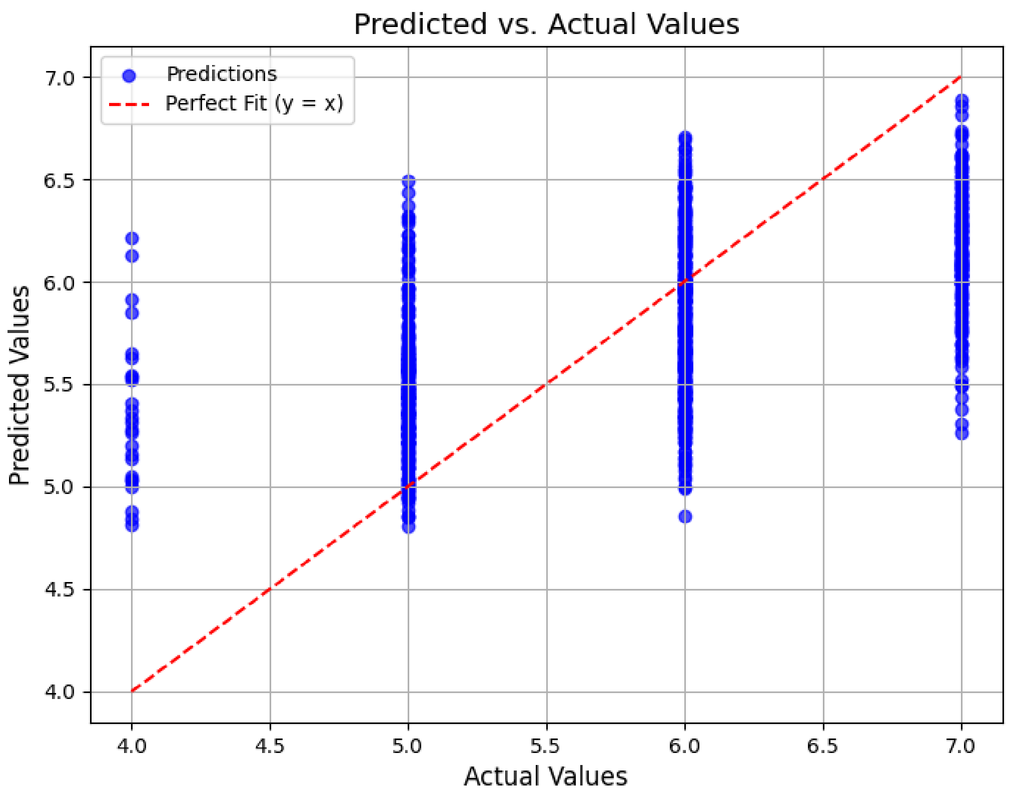
Hyperparameter comparison

All the hyperparameter tuning is done using our neural network. The table below shows the comparison using three metrics; MAPE, MAE, MSE.

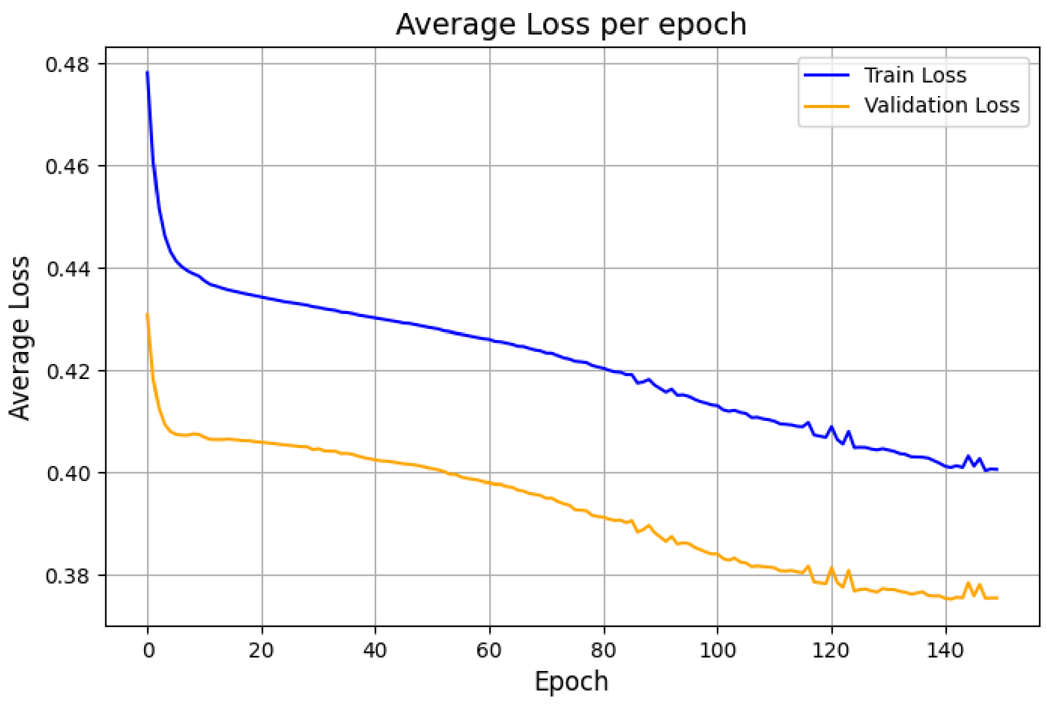
	Number of layers	Layer Structure	Num epochs	Learning Rate	Momentum	Activation function	MAPE	MAE	MSE
1	4	[13, 32, 8, 1]	100	0.001	0	ReLU	0.088	0.502	0.390
2	3	[13, 10, 1]	100	0.001	0	ReLU	0.089	0.503	0.393
3	8	[13, 128, 64, 32, 16, 8, 4, 1]	200	0.001	0.2	ReLU	0.106	0.578	0.591
4	4	[13, 64, 8, 1]	150	0.001	0.1	ReLU	0.088	0.502	0.392
5	4	[13, 32, 4, 1]	100	0.001	0	tanh	0.092	0.513	0.392
6	6	[13, 16, 16, 8, 8, 1]	100	0.001	0.1	tanh	0.091	0.515	0.389
7	4	[13, 16, 8, 1]	100	0.01	0.1	ReLU	0.089	0.505	0.394
8	5	[13, 128, 32, 4, 1]	100	0.01	0.3	ReLU	0.111	0.622	0.581
9	5	[13, 64, 16, 4, 1]	150	0.001	0.05	sigmoid	0.111	0.624	0.580
10	4	[13, 20, 10, 1]	150	0.001	0.1	ReLU	0.087	0.495	0.378
11	5	[13, 256, 128, 16, 1]	200	0.001	0.5	ReLU	0.111	0.621	0.581
12	3	[13, 50, 1]	200	0.001	0	ReLU	0.088	0.497	0.379
13	9	[13, 70, 60, 50, 40, 30, 20, 10, 1]	100	0.01	0	ReLU	0.086	0.488	0.388

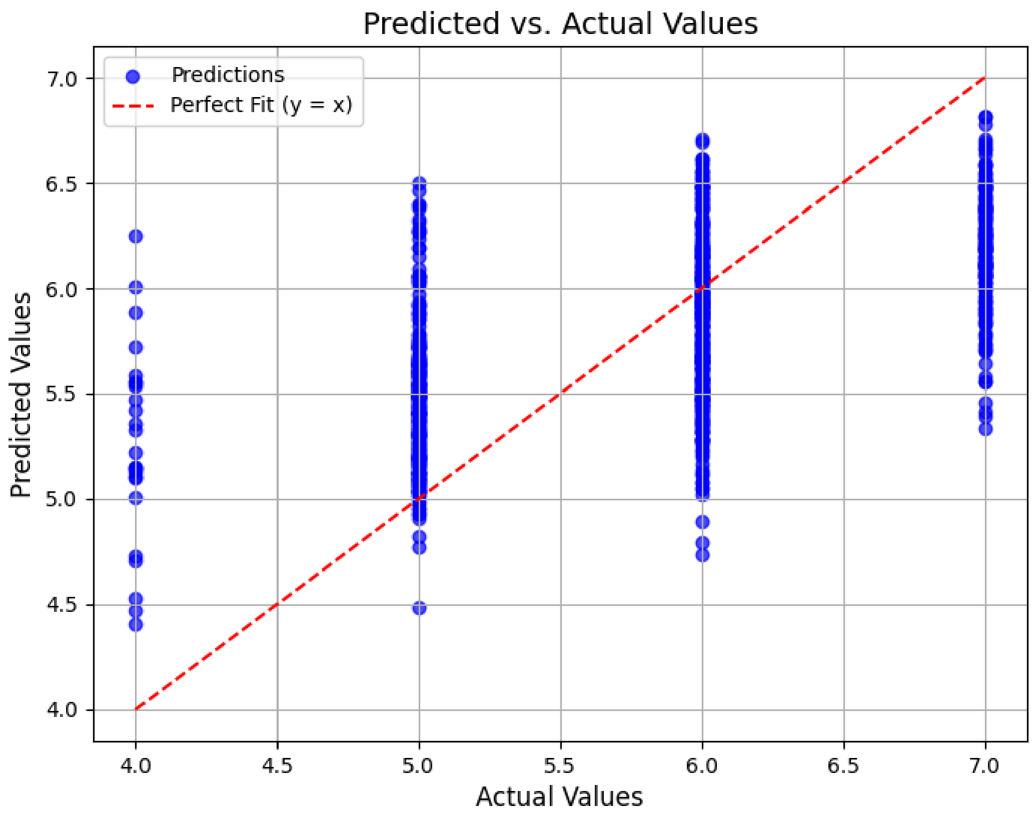
Let us consider the MAPE metrics as the *best* one and use it for the model comparison. From table details above, the best models in this setting are #1, #10 and #13. One similarity we can see is the activation function. Besides the three activation functions used during the hyperparameter tuning, the ReLU performs the best. The main reason is probably the fact that we face the regression task and not the classification one, so there is no need to strict the values space to smaller ones as both *sigmoid* and *tanh* do.

Next to the choice of an activation function, networks with a big number of neurons per one (mostly the first) layer also perform poorly. The number of epochs in general works as “the more the best” because there is more time to learn some patterns. In this case, we use 100 - 200 epochs for training. The momentum has not any significant effect on the final model during those epochs, and it is hard to say if the learning rate has or not. Finally, the last neural net has many more layers than the previous ones and also performs well (in case of those parameters). It is hard to say why. Below, we provide scatter plots of the predictions and also the development of loss during training for the three best performing neural networks.

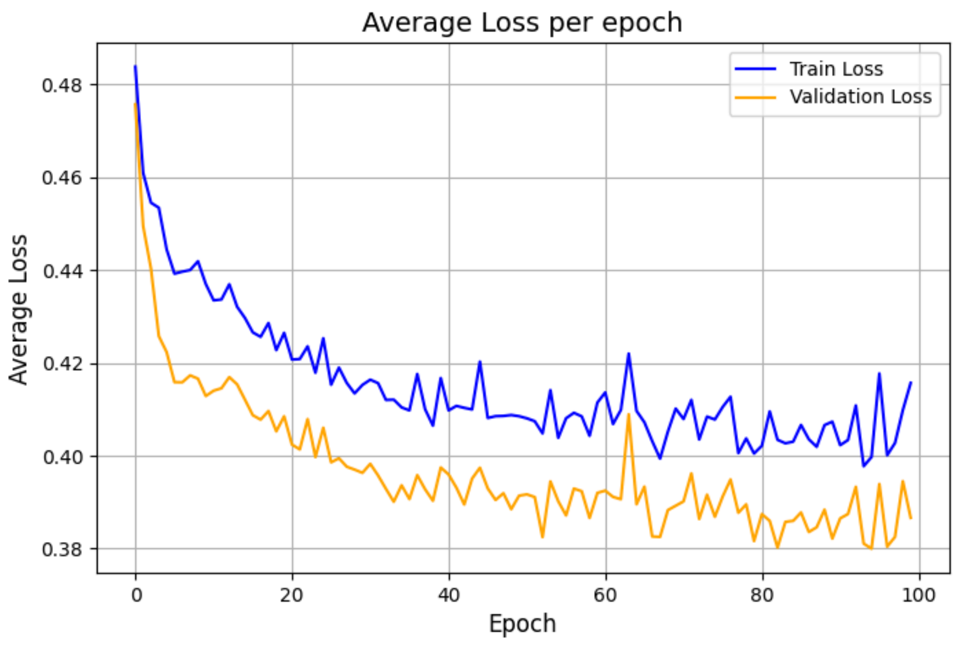


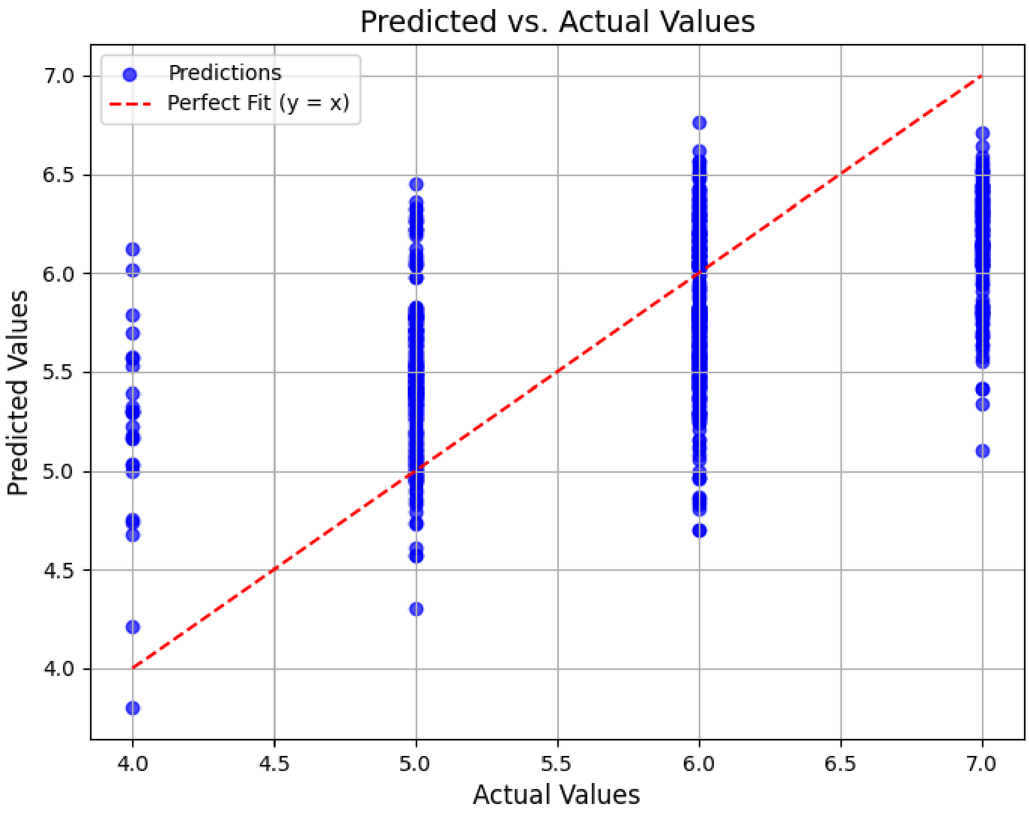
Those graphs correspond to the first neural network setting. As we can see, the loss is still decreasing, so training with more epochs could improve the final precision. From the scatter plot we can see that the majority of predictions are biased around 5 to 6.5.





Those two graphs correspond to the 10th neural network. The loss functions seem converged at the very last epochs, but still, few more epochs could improve the final result, or at least ensure the convergence. The scatter plot is almost the same as for the previous neural network; it again struggles between 5 and 6.5.



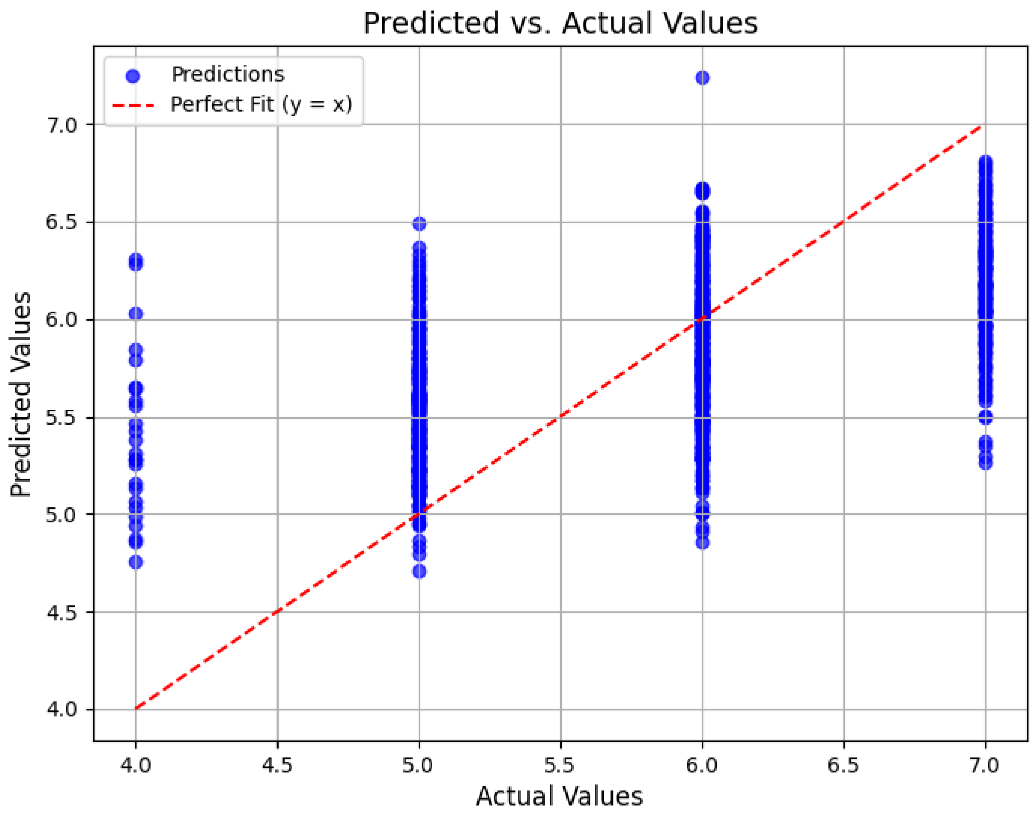


The scatter plot for the last 13th neural network is, again, similar to the previous two. But the big difference is in the loss over epochs. This one is much more volatile. It seems necessary to do some improvement to this neural net to get more stable training. The main problem is probably the bigger learning rate, which ensures this kind of oscillation.

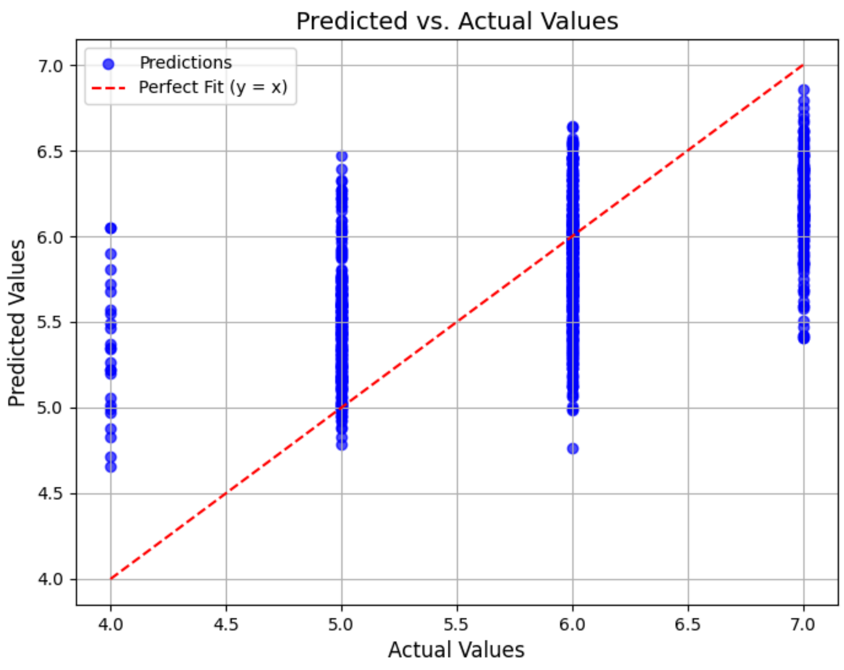
Model comparison

In this section, we will discuss all three different models we used on the same dataset.

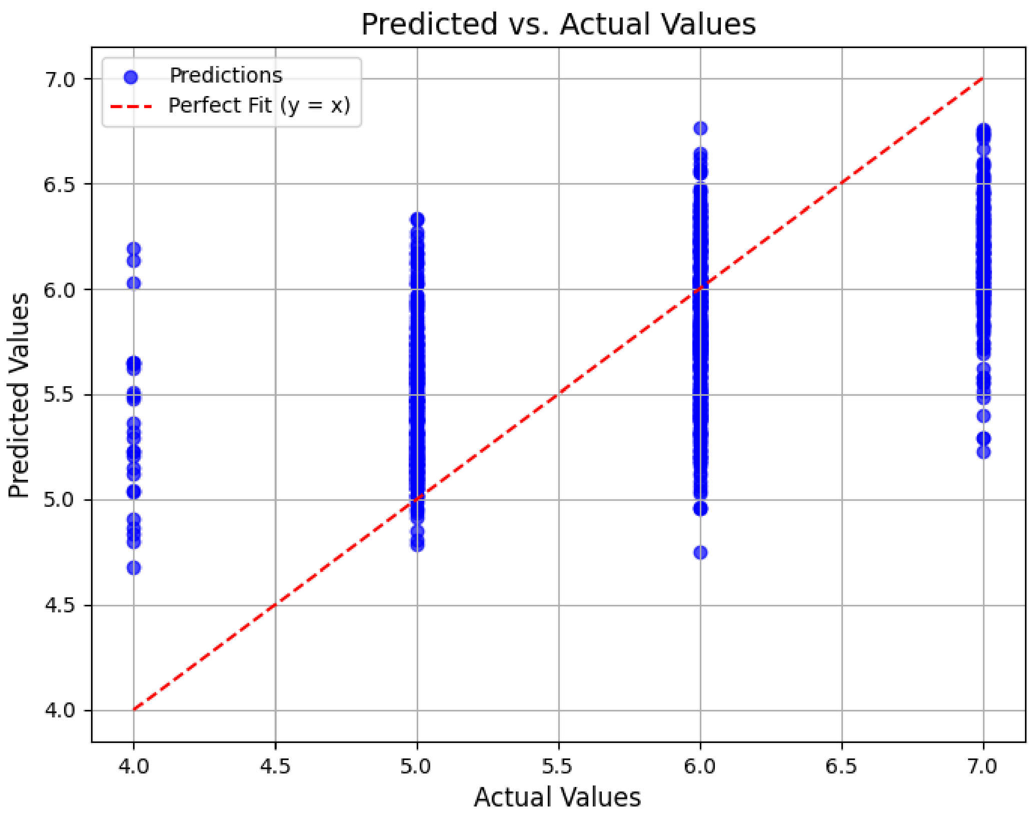
	MSE	MAE	MAPE
MLR	0.407	0.515	0.091
our-NN	0.388	0.499	0.088
pytorch-NN	0.399	0.509	0.089



Scatter plot of the mlr model.



Scatter plot of neural network with implemented back propagation.



Scatter plot of the pytorch neural network.

As we can see, all models perform really similarly. Surprisingly, our neural network perform best from those three models. I would expect that the mlr would be the worst, because it is still **linear** model. Both neural networks, our and pytorch, has the same number of layers, number of neurons and same hyperparameters during training. It corresponds to the first neural network from the *hyperparameter* table. Both models are quite close in their predictions. In this specific case our neural network is better, but I think with the second run (and thus different weight init) it could be different. The main problem in our case is the dataset. There are definitely not enough data, especially on both sides of the quality space - really good wines and really bad wines. The few samples the original dataset had were removed during the outliers detection. Regardless this insufficiency, also more epochs for training could help little bit. But in the end, I am not sure how should the neural net looks like to give much better predictions.

Github link: https://github.com/kaminno/NEC_neural_network